Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Sabrina Wilske

Software-Projekt 2 2013/14

VAK 03-BA-901.02

Architekturbeschreibung

IT_R3V0LUT10N

Sebastian Bredehöft	sbrede@tzi.de	2751589
Patrick Damrow	damsen@tzi.de	2056170
Tobias Dellert	tode@tzi.de	2936941
Tim Ellhoff	tellhoff@tzi.de	2520913
Daniel Pupat	dpupat@tzi.de	2703053

Abgabe: 22. Dezember 2013 – Version 1.1 vom 08.12.2013

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Version und Änderungsgeschichte

Version	Datum	Änderungen
1.0	25.11.2013	Dokumentvorlage als initiale Fassung kopiert
1.1	08.12.2013	Einflussfaktoren

1 Einführung

1.1 Zweck

Was ist der Zweck dieser Architekturbeschreibung? Wer sind die LeserInnen?

1.2 Status

1.3 Definitionen, Akronyme und Abkürzungen

1.4 Referenzen

1.5 Übersicht über das Dokument

2 Globale Analyse

2.1 Einflussfaktoren

Die Einflussfaktoren werden im Folgenden unterteilt in:

- Organisatorische Faktoren
- Technische Faktoren
- Produktfaktoren

2.1.1 Organisatorische Faktoren

Tabelle 1: Organisatorische Faktoren

O1	**Time-To-Market**
O2	**Auslieferung von Produktfunktionen**
O3	**Budget**
O4	**Kenntnisse in Java und Android**
O5	**Kenntnisse in J-Unit**
O6	**Anzahl der Entwickler**

O1	**Time-To-Market**
Faktor	Auslieferungsdatum 23.02.2014
Flexibilität und Veränderlichkeit	Die Deadline kann nicht verändert werden.
Auswirkungen	Die Software muss zum Abgabedatum lauffähig sein.

O2	**Auslieferung von Produktfunktionen**
Faktor	Alle Mindestanforderungen
Flexibilität und Veränderlichkeit	Es müssen alle Mindestanforderungen erfüllt sein; sie sind jedoch vom Kunden oder beim Verlassen eines Gruppenmitglieds veränderbar.
Auswirkungen	Architektur muss alle Mindestanforderungen abdecken; es muss darauf geachtet werden, dass diese sich im Verlauf noch ändern.

O3	**Budget**
Faktor	Kein finanzielles Budget
Flexibilität und Veränderlichkeit	Es werden keine finanziellen Unterstützungen für das Produkt geben.
Auswirkungen	Es können keine kostenpflichtigen Dienste in Anspruch genommen werden.

O4	**Kenntnisse in Java und Android**
Faktor	Kenntnisse der Entwickler in Java und Android
Flexibilität und Veränderlichkeit	Kenntnisse sind nicht flexibel, es muss in Java programmiert werden und über Smartphone laufen. Die Kenntnisse können sich im Laufe ändern, z.B. durch neue Erfahrungen und neu erworbene Kenntnisse.
Auswirkungen	Bei wenig Kenntnissen muss mehr Zeit eingeplant werden, um sich diese anzueignen.

O5	**Kenntnisse in J-Unit**
Faktor	Kenntnisse in J-Unit Tests
Flexibilität und Veränderlichkeit	Da Tests mit J-Unit gefordert werden, sind diese nicht verhandelbar oder flexibel.
Auswirkungen	Bei unzureichenden Tests kann es später beim Programm zu Problemen kommen, da Fehler spät oder gar nicht erkannt werden.

O6	**Anzahl der Entwickler**
Faktor	Die Anzahl der Entwickler
Flexibilität und Veränderlichkeit	Es können keine neuen Gruppenmitglieder dazukommen, es können aber jederzeit Gruppenmitglieder wegfallen.
Auswirkungen	Wenn Gruppenmitglieder wegfallen, müssen die restlichen Mitglieder mehr Arbeit und mehr Zeit einplanen. Auch müssen Projektplan und Architektur neu angepasst werden.

2.1.2 Technische Faktoren

Tabelle 2: Technische Faktoren

T1	**Software funktioniert unter Windows, Linux und MacOS**
T2	**Software funktioniert als App(Andriod 2.3 oder höher)**
T3	**SQL-Datenbank**
T4	**Mehrere parallele Nutzer**
T5	**Client-Server System**
T6	**Benutzerschnittstelle**
T7	**Implementierungssprache Java**
T8	**Beschränkungsfreiheit für Fremdbibliotheken**

T1	**Software funktioniert unter Windows, Linux und MacOS**
Faktor	Die Software muss auf den Betriebssystemen Windows, Linux und MacOS laufen.
Flexibilität und Veränderlichkeit	nicht flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Entwickler müssen sich mit allen Betriebsprogrammen befassen und sichergehen, dass es auf allen funktioniert.

T2	**Software funktioniert als App (Andriod 2.3 oder höher).**
Faktor	Die Software muss als Android App auf einem Smartphone laufen.
Flexibilität und Veränderlichkeit	nicht flexibel, da dies zu den Mindestanforderungen gehört. Veränderungen können jederzeit vom Kunden vorgenommen werden.
Auswirkungen	Die Software muss wie gefordert als App auf einem Android-Smartphone laufen.

T3	**SQL-Datenbank**
Faktor	Software läuft über eine relationale Datenbank.
Flexibilität und Veränderlichkeit	Flexibel, jedoch muss eine Datenbank mit SQL oder SQL-ähnlichen Abfragen verwendet werden.
Auswirkungen	Es muss eine relationale Datenbank für die serverseitige Persistenz benutzt werden. Es muss eine Datenbank mit SQL oder SQL-ähnlichen abfragen verwendet werden.

T4	**Mehrere parallele Nutzer**
Faktor	Es greifen mehrere Nutzer zur gleichen Zeit auf die Software zu.
Flexibilität und Veränderlichkeit	Es ist uns überlassen, wie viele Nutzer zur gleichen Zeit auf das System zugreifen dürfen.
Auswirkungen	Die Software muss darauf ausgelegt sein, mehrere Nutzer zur gleichen Zeit zu verwalten.

T5	**Client-Server System**
Faktor	Die Software arbeitet über ein Client-Server System.
Flexibilität und Veränderlichkeit	Da wir übers Internet auf den Server zugreifen müssen, ist es notwendig, ein Server-Client System zu verwenden.
Auswirkungen	Die Implementierung wird in Server und Client aufgeteilt (siehe 3). Übers Internet werden die Daten zwischen Server und Client ausgetauscht.

T6	**Benutzerschnittstelle**
Faktor	Es sollte eine übersichtliche und ansprechende GUI geben.
Flexibilität und Veränderlichkeit	Die Gestaltung der GUI ist uns überlassen.
Auswirkungen	Für eine benutzerfreundliche Gestaltung sind gute Kenntnisse in XHTML notwendig.

T7	**Implementierungssprache Java**
Faktor	Die Software muss in Java 5 oder höher geschrieben werden.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies zu den Mindestanforderungen gehört.
Auswirkungen	Die Software muss in Java geschrieben werden, daher müssen alle Entwickler diese Sprache beherrschen.

T8	**Beschränkungsfreiheit für Fremdbibliotheken**
Faktor	Fremdbibliotheken dürfen für den Einsatz in Forschung. und Lehre keine Beschränkungen aufweisen.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies zu den Mindestanforderungen gehört.
Auswirkungen	Es darf keine Software oder Bibliothek verwendet werden, die kostenpflichtig ist.

2.1.3 Produktfaktoren

Tabelle 3: Produktfaktoren

P1	**Mindestanforderung**
P2	**Performanz**
P3	**Benutzerrechte**
P4	**Fehlererkennung**

P1	**Mindestanforderung**
Faktor	Das Produkt muss alle Mindestanforderungen enthalten.
Flexibilität und Veränderlichkeit	Alle Anforderungen müssen zum Bestehen erfüllt werden. Die Anforderungen können vom Kunden oder Dozenten verändert werden oder die Anforderungen werden bei einem Austritt eines Mitglieds verringert.
Auswirkungen	Es müssen alle Mindestanforderungen implementiert werden.

P2	**Performanz**
Faktor	Möglichst schnelle Ausführungszeiten
Flexibilität und Veränderlichkeit	Flexibel, da nichts davon in den Mindestanforderungen steht.
Auswirkungen	Es sollte bei der Implementierung auf einen schnellen Datenaustausch zwischen Server und Client geachtet werden.

P3	**Benutzerrechte**
Faktor	Es gibt verschiedene Benutzer mit unterschiedlichen Rechten.
Flexibilität und Veränderlichkeit	Nicht flexibel, da dies vom Kunden gefordert wird.
Auswirkungen	Es müssen unterschiedliche Benutzer implementiert werden, die unterschiedliche Rechte haben und diese auch nicht überschreiten dürfen.

P4	**Fehlererkennung**
Faktor	Fehler sollten von der Software erkannt werden und entsprechend behandelt werden.
Flexibilität und Veränderlichkeit	Flexibel, da dies nicht ausdrücklich vom Kunden gefordert wird.
Auswirkungen	Fehler müssen erkannt und durch entsprechende Exceptions korrigiert werden. Die Software sollte weiter laufen.

2.2 Probleme und Strategien

Folgenden Probleme haben wir identifiziert:

Nummer	**Faktoren**
1	Zeitprobleme
2	Mangelnde Kenntnisse in Java
3	Mangelnde Kenntnisse in Android
4	Mangelnde Kenntnisse von Datenbanksystemen
5	Unzureichende Softwaretests
6	Ausfall eines Gruppenmitglieds
7	Mehrere parallele Nutzer
8	Performanz
9	unterschiedliche Benutzerrechte

Diese versuchen wir mit folgenden Strategien zu überbrücken:

1 Zeitprobleme
Es gibt einen festgesetzten Abgabetermin, der eingehalten werden muss.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java und Android • O5 Kenntnisse in J-Unit • O6 Anzahl der Entwickler • P1 Mindestanforderungen
Lösung • Strategie 1: Modularisierung für paralleles Arbeiten Durch Modularisierung können mehrere Entwickler zur gleichen Zeit am Projekt arbeiten und die Module unabhängig voneinander implementieren. Diese werden dann später zusammengesetzt. • Strategie 2: Bibliotheken Benutzen Es werden bereits vorhandene Java Bibliotheken verwendet; dies spart Zeit, da man dann nicht alles neu schreiben muss. Es werden beide Strategien verwendet.

2 Mangelnde Kenntnisse in Java
Es werden Vorkenntnisse in Java vorausgesetzt; ohne diese könnte es zu großen Problemen kommen, da ohne ausreichende Kenntnisse das Programm nicht realisiert werden kann.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java und Android • O6 Anzahl der Entwickler • T1 Software funktioniert unter Windows, Linux und MacOS • T5 Client-Server System • T7 Implementierungssprache Java • P1 Mindestanforderungen
Lösung • Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt. Wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module anrichten. • Strategie 2: Aufteilen in Server und Client Die Implementierung wird unter den Entwicklern so aufgeteilt, dass ein Teil den Client und der andere Teil den Server macht; so müssen sich die Gruppenmitglieder nicht Kenntnisse in beiden Bereichen aneignen. Es werden beide Strategien verwendet.

3 Mangelnde Kenntnisse in Android
Es werden Kenntnisse in Android vorausgesetzt, da eine App entwickelt werden muss.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O4 Kenntnisse in Java und Android • O6 Anzahl der Entwickler • T2 Software funktioniert als App(Andriod 2.3 oder höher) • T5 Client-Server System • T6 Benutzerschnittstelle • T7 Implementierungssprache Java • P1 Mindestanforderungen
Lösung • Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt. Wenn ein Gruppenmitglied nicht genügend Kenntnisse besitzt, kann dieses Modul von einem anderen Mitglied neu erstellt werden und der inkompetente Entwickler kann keinen Schaden auf andere Module anrichten. • Strategie 2: Bearbeitung von Gruppenmitgliedern mit Android-Erfahrung Wir werden die Implementierung einem Gruppenmitglied überlassen, das bereits Erfahrung mit Android hat. So müssen sich die anderen nicht in Android einarbeiten und können sich bei Fragen an eben diesen wenden. Es werden beide Strategien verfolgt; sollte das Gruppenmitglied mit Android zeitlich oder fachlich nicht klarkommen, wird sich ein weiteres Gruppenmitglied mit Android beschäftigen.

4 Mangelnde Kenntnisse in Datenbanksystemen
Es werden Kenntnisse in Datenbanksystemen vorausgesetzt, da wir für die Bibliothek eine Datenbank verwenden. Dabei werden SQL- oder SQL-ähnliche Abfragen verwendet und entsprechende Kenntnisse verlangt.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O7 Kenntnisse in Datenbanksystemen • O6 Anzahl der Entwickler • T5 Client-Server System • T7 Implementierungssprache Java
Lösung • Strategie 1: Bearbeitung von Gruppenmitgliedern mit Erfahrung in Datenbanksystemen Wir werden die Implementierung Gruppenmitgliedern überlassen, die bereits Erfahrung mit Datenbanksystemen haben. So müssen sich die anderen nicht in Datenbanksystemen einarbeiten und können sich bei Fragen an diese wenden.

5 Unzureichende Softwaretests
Es werden genügend Tests benötigt, welche Module und Komponenten testen, ob diese funktionieren.
Einflussfaktoren • O1 Time-To-Market • O5 Kenntnisse in J-Unit • T7 Implementierungssprache Java • P1 Mindestanforderungen • P2 Performanz • P3 Benutzerrechte • P4 Fehlererkennung
Lösung • Strategie 1: Modularisierung Es werden Tests für die jeweilig implementierten Module geschrieben. Ziel ist es, zu prüfen, ob diese ihren Zweck erfüllen und danach werden Module zusammen getestet.

6 Ausfall eines Gruppenmitglieds
Es kann jederzeit ein Gruppenmitglied aus der Gruppe austreten oder durch Krankheit etc. für eine gewisse Zeit ausfallen.
Einflussfaktoren • O1 Time-To-Market • O2 Auslieferung von Produktfunktionen • O6 Anzahl der Entwickler • P1 Mindestanforderungen
Lösung • Strategie 1: Modularisierung Der Code wird in verschiedene Module aufgeteilt, die von einem Entwickler bearbeitet werden. Wenn nun ein Entwickler ausfällt, kann ein Modul von einem anderen Entwickler übernommen werden.

7 Mehrere parallele Nutzer
Es greifen mehrere Nutzer zur gleichen Zeit auf das System zu, auf das der Server antworten muss. Dabei soll der Server die Daten nicht an alle Clients senden.
Einflussfaktoren • O1 Time-To-Market • O3 Budget • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Mindestanforderungen • P2 Performanz • P3 Benutzerrechte
Lösung • Strategie 1: Thread Die Clients bekommen jeweils einen Thread. Somit können sie zeitgleich auf den Server zugreifen und bekommen nur ihre Daten zurück.

8 Performanz
Die Software sollte kurze Ausführungszeiten haben. Dabei ist zu beachten, dass die Software/App auch auf Geräten mit geringer Leistung schnell und problemlos läuft.
Einflussfaktoren • O1 Time-To-Market • T1 Software funktioniert unter Windows, Linux und MacOS • T2 Software funktioniert als App(Android 2.3 oder höher) • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Mindestanforderungen • P2 Performanz • P3 Benutzerrechte
Lösung • Strategie 1: Code effizient schreiben Den Code effizient schreiben, damit die Software kurze Ausführungszeiten hat.

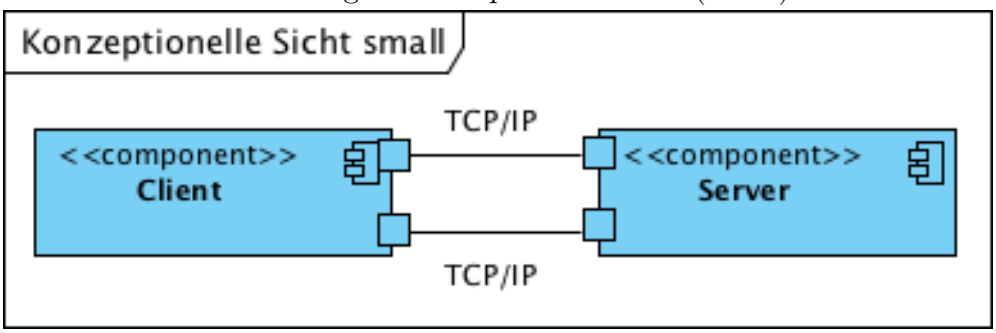
9 unterschiedliche Benutzerrechte
Die Software hat unterschiedliche Benutzer, welche unterschiedliche Rechte besitzen und diese müssen unterschieden werden.
Einflussfaktoren • O1 Time-To-Market • T3 SQL-Datenbank • T4 Mehrere parallele Nutzer • P1 Mindestanforderungen • P3 Benutzerrechte
Lösung • Strategie 1: Identifikation durch Group Id In der Datenbank wird eine Group Id eingefügt, die dann die verschiedenen Nutzer speichert. Über diese werden dann die verschiedenen Rechte geregelt.

3 Konzeptionelle Sicht

Wir haben mithilfe von UML-Diagrammen die konzeptionelle Sicht realisiert. Im Folgenden werden die einzelnen Diagramme aufgezeigt und beschrieben und in nachfolgenden Sichten zusätzlich verfeinert und konkretisiert.

3.1 Überblick

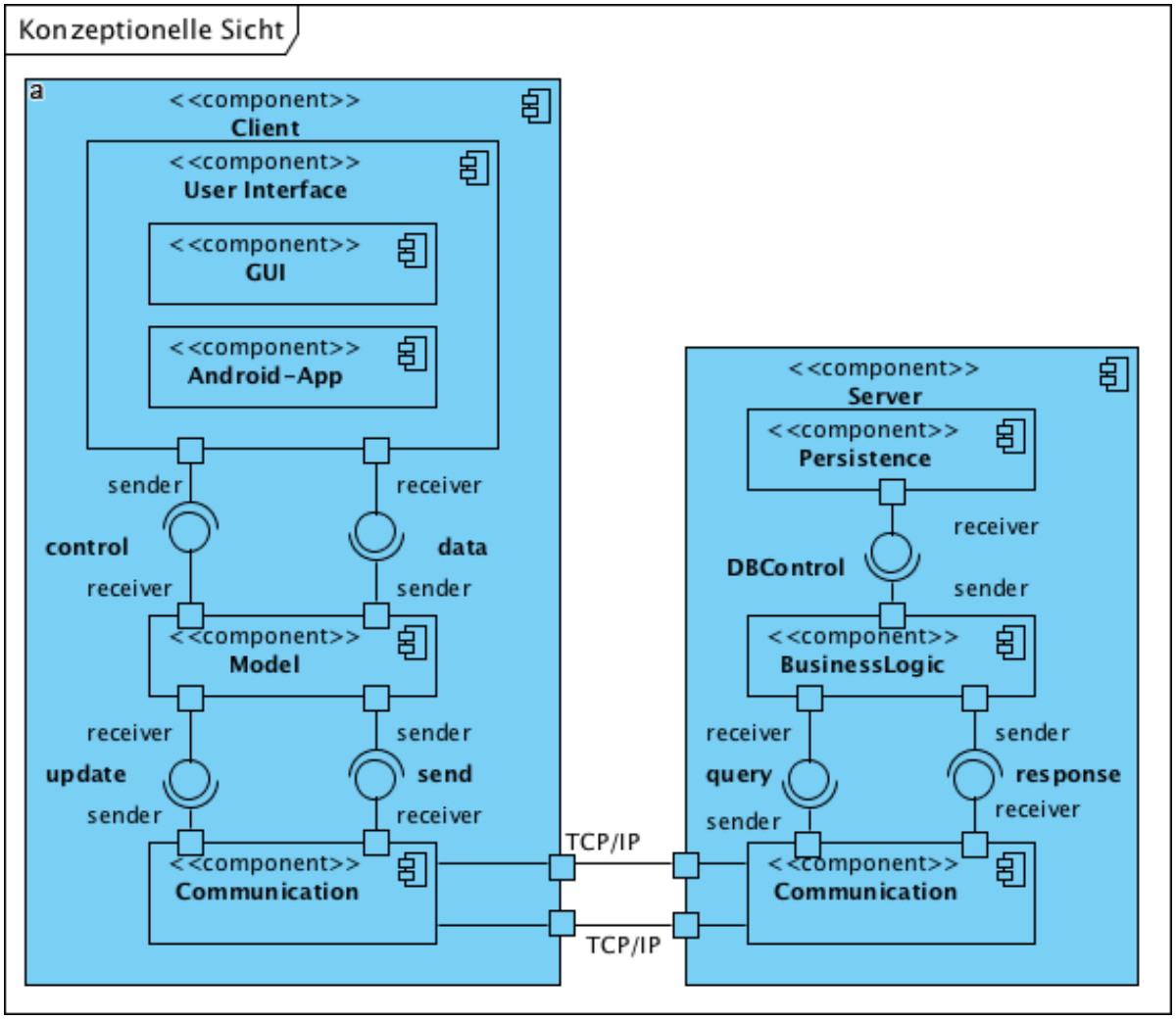
Abbildung 1: Konzeptionelle Sicht (Klein)



Unsere Architektur besteht aus zwei grundlegenden Komponenten, der Serverkomponente und der Clientkomponente. Diese Komponenten beinhalten wiederum weitere Komponenten. Auf der einen Seite haben wir unsere Serverkomponente, die alle benötigten Daten der Medien, der Nutzer und der Ausleihvorgänge der Bibliothek speichert.

Auf der anderen Seite, der Clientkomponente, muss zwischen zwei Teilen unterschieden werden. Einmal dem GUI-Client, welcher sich in erster Linie an die Bibliothekare richtet und der mobile Android-Client, der sich ausschließlich an die Leser richtet.
Der GUI-Client stellt für die Bibliothekare alle benötigten Funktionen bereit, um eine Bibliothek zu verwalten. Der Android-Client ermöglicht dem Leser, sich Mediendetails, Ausleihstatus, seine eigene Ausleihhistorie sowie persönliche Daten anzeigen zu lassen. Des Weiteren kann der Leser sich mittels der App Bücher zur Ausleihe vormerken und Informationen über die Bibliothek abrufen.

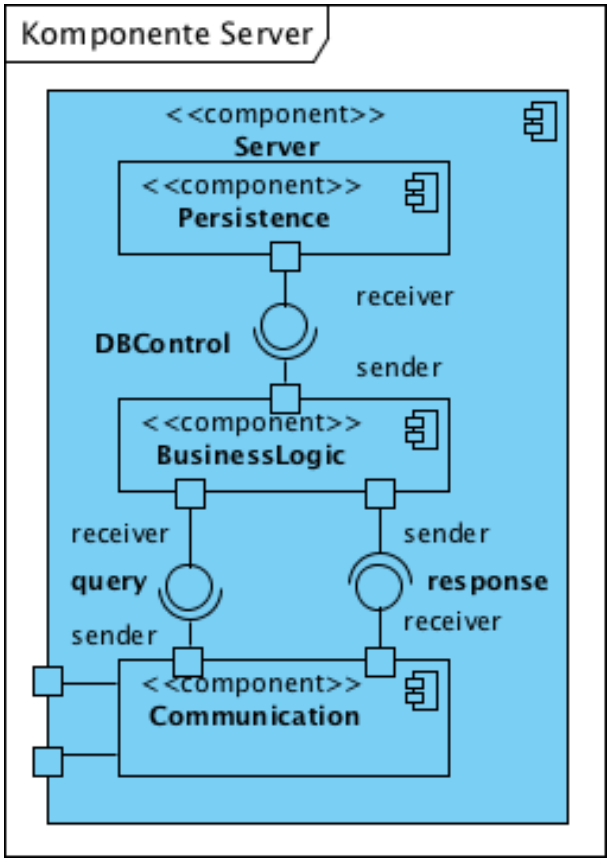
Abbildung 2: Konzeptionelle Sicht



Als Architekturstil verwenden wir das Model-View-Controller-Pattern.

3.2 Serverkomponente

Abbildung 3: Konzeptionelle Sicht Server



Die Serverkomponente besteht aus insgesamt drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

  Die Komponente `Communication` nimmt Anfragen des Clients entgegen und leitet sie an die Komponente `BusinessLogic` weiter, wo die Anfragen verarbeitet werden und sendet die Ergebnisse zurück an den Client.

- BusinessLogic

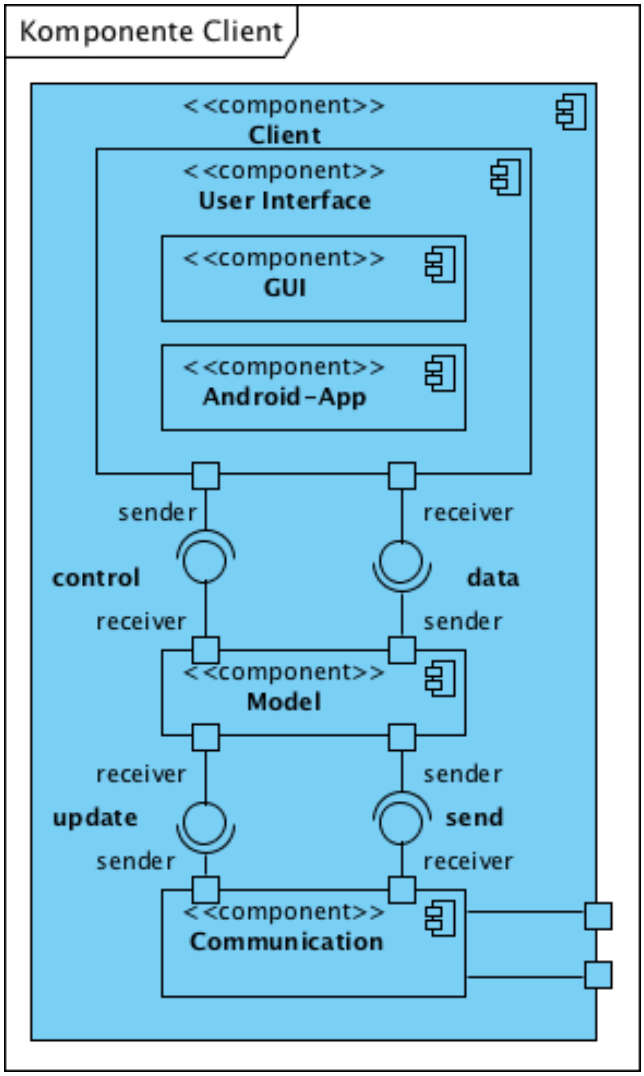
  Die Komponente `BusinessLogic` dient zum Verarbeiten der Anfragen und leitet diese verarbeiteten Anfragen dann an die Komponente `Persistence` weiter.

- Persistence

  Die Komponente `Persistence` ist die Schnittstelle zur Datenbank. Über das Interface `DBControl` werden die verarbeiteten Anfragen von der Komponente `BusinessLogic` empfangen und in Datenbankabfragen umgewandelt, welche dann von der Datenbank entgegen genommen werden.

3.3 Clientkomponente

Abbildung 4: Konzeptionelle Sicht Client



Die Clientkomponente besteht so wie die Serverkomponente aus drei Teilkomponenten, welche sich wie folgt aufgliedern:

- Communication

  Die Komponente `Communication` sendet Anfragen des Clients an den Server, welche dort verarbeitet werden und nimmt die Ergebnisse entgegen, um diese an die Komponente `Model` zu übergeben, wo die Ergebnisse der Anfrage weiter verarbeitet werden.

- Model

  Die Komponente `Model` nimmt Ergebnisse von der Komponente `Communication`

entgegen und schickt diese an die Komponente `User Interface`.

- User Interface

  Die Komponente `User Interface` muss in zwei unterschiedliche Komponenten zerlegt werden:

  – GUI

    Die GUI richtet sich in erster Linie an Bibliothekare.

  – Android-App

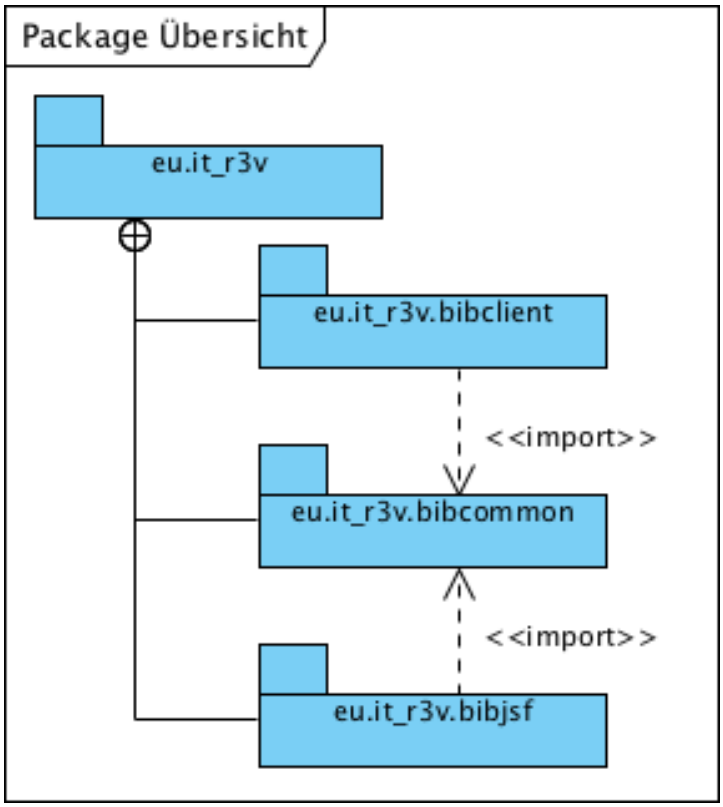
    Die Android-App richtet sich in erster Linie an die Nutzer/Leser.

4 Modulsicht

4.1 Pakete

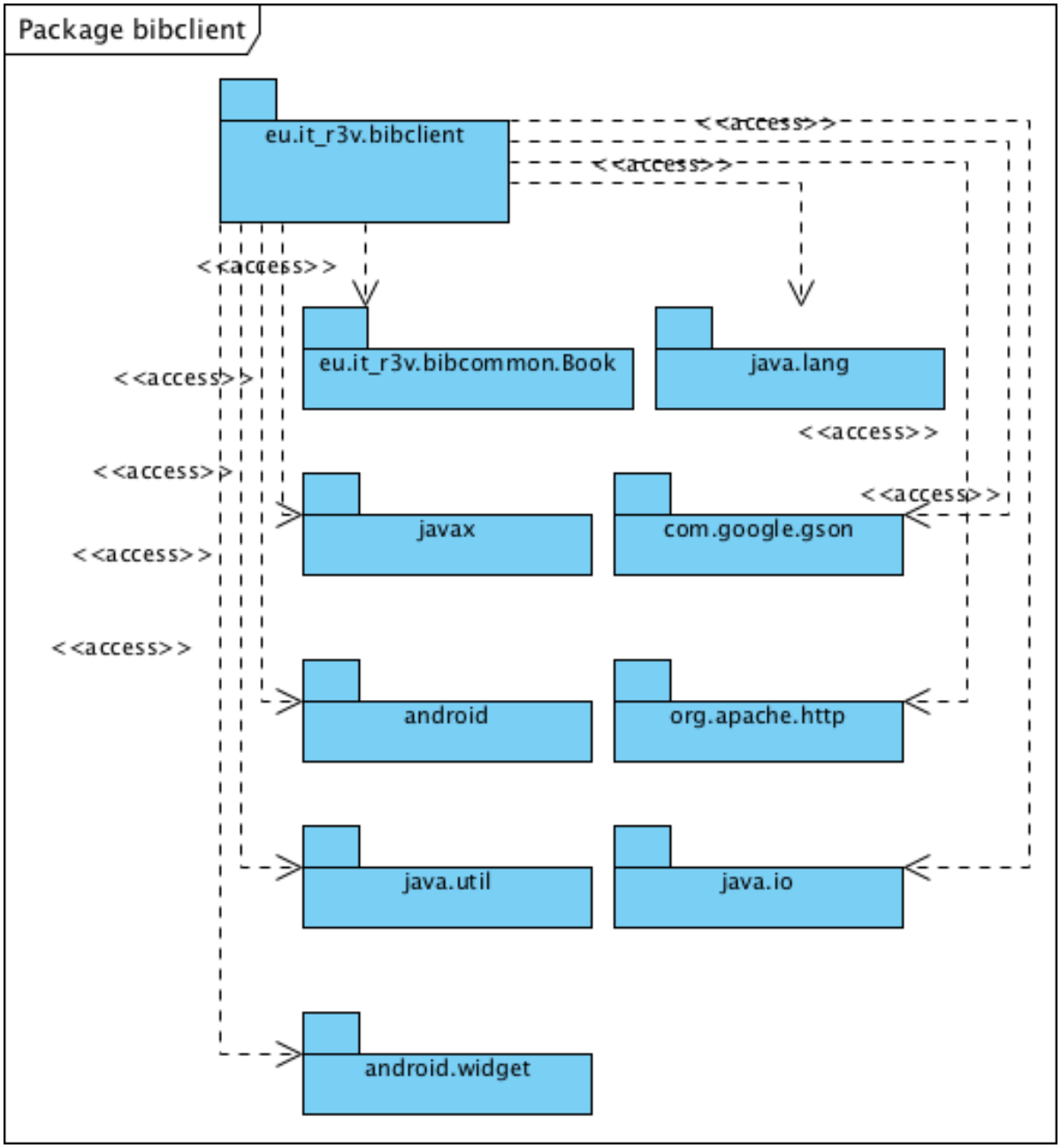
Wir haben ein Hauptpaket `eu.it_r3v`, in dem sich weitere Unterpakete befinden. Diese dienen der Bündelung gemeinsamer Quellcodedateien.

Abbildung 5: Pakete Übersicht



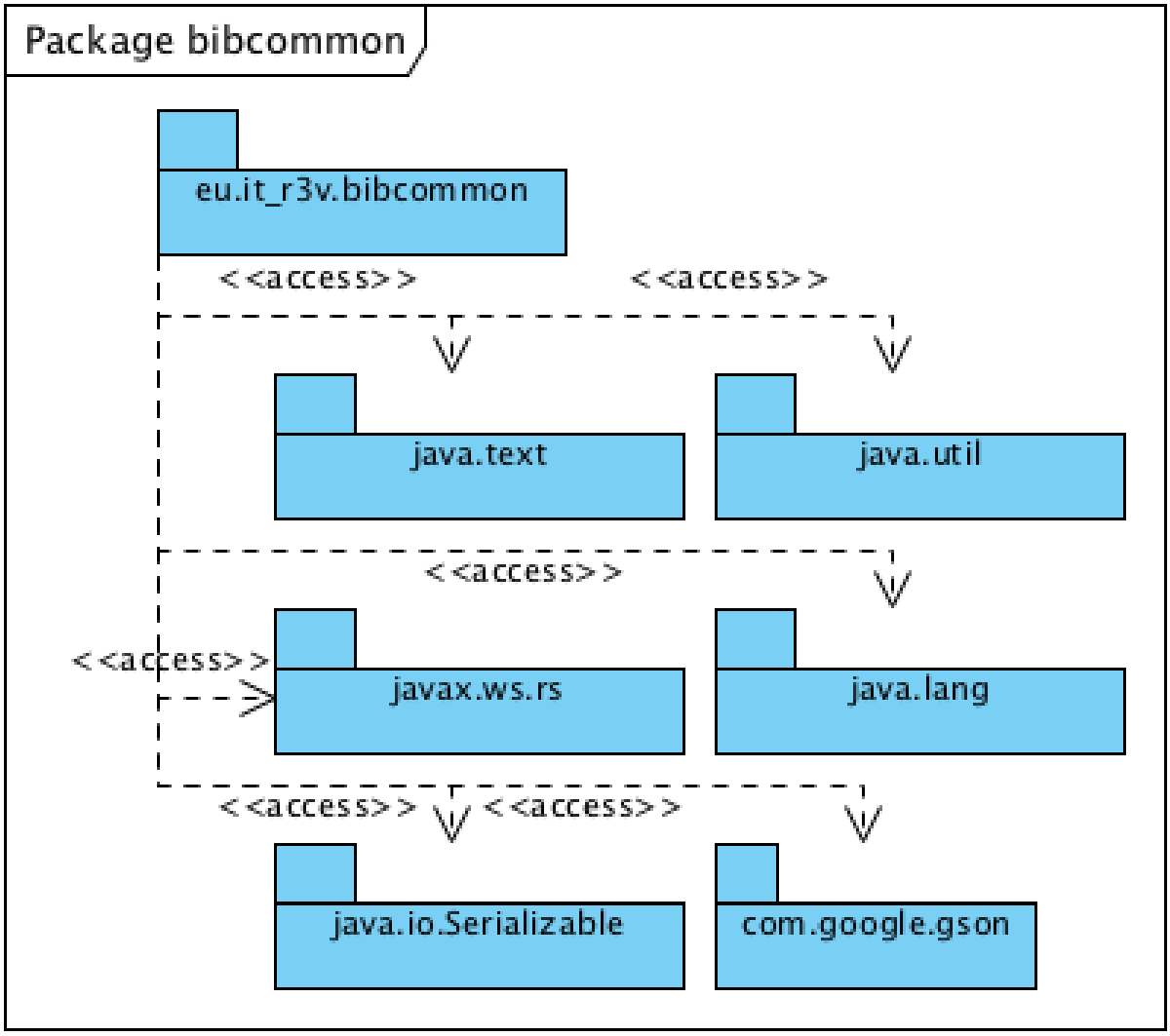
4.1.1 Paket `bibclient`

Abbildung 6: Paket `bibclient`



4.1.2 Pakete `bibcommon`

Abbildung 7: Paket `bibcommon`



4.1.3 Paket `bibjsf`

Abbildung 8: Paketübersicht `bibjsf`

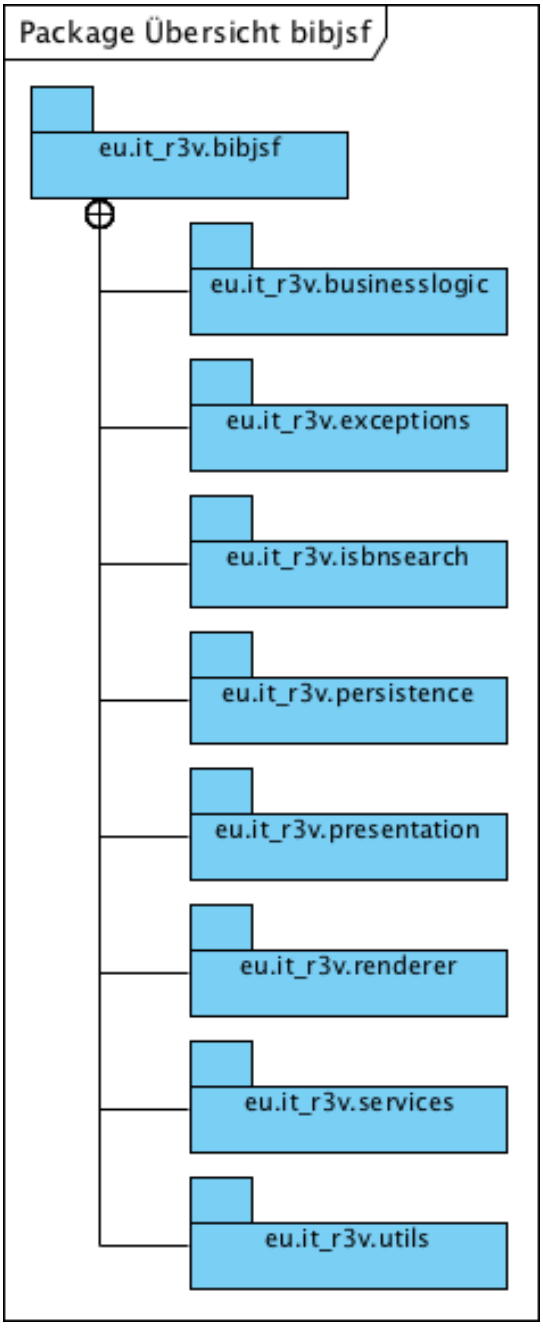
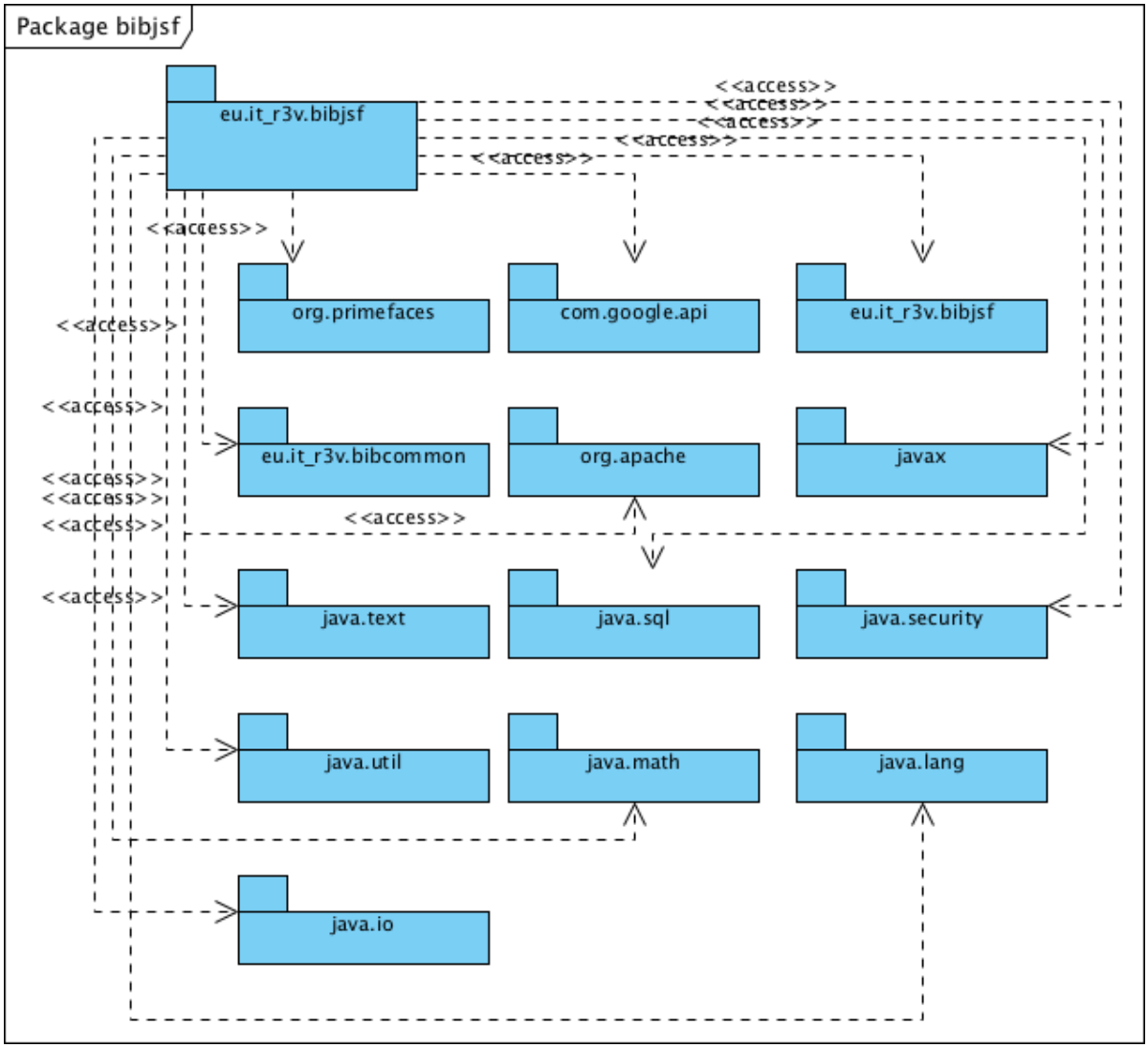


Abbildung 9: Paket bibjsf



Diese Sicht beschreibt den statischen Aufbau des Systems mit Hilfe von Modulen, Subsystemen, Schichten und Schnittstellen. Diese Sicht ist hierarchisch, d.h. Module werden in Teilmodule zerlegt. Die Zerlegung endet bei Modulen, die ein klar umrissenes Arbeitspaket für eine Person darstellen und in einer Kalenderwoche implementiert werden können. Die Modulbeschreibung der Blätter dieser Hierarchie muss genau genug und ausreichend sein, um das Modul implementieren zu können.

Die Modulsicht wird durch UML-Paket- und Klassendiagramme visualisiert.

Die Module werden durch ihre Schnittstellen beschrieben. Die Schnittstelle eines Moduls M ist die Menge aller Annahmen, die andere Module über M machen dürfen, bzw. jene Annahmen, die M über seine verwendeten Module macht (bzw. seine Umgebung, wozu auch Speicher, Laufzeit etc. gehören). Konkrete Implementierungen dieser Schnittstellen

sind das Geheimnis des Moduls und können vom Programmierer festgelegt werden. Sie sollen hier dementsprechend nicht beschrieben werden.

Die Diagramme der Modulsicht sollten die zur Schnittstelle gehörenden Methoden enthalten. Die Beschreibung der einzelnen Methoden (im Sinne der Schnittstellenbeschreibung) geschieht allerdings per Javadoc im zugehörigen Quelltext. Das bedeutet, dass Ihr für alle Eure Module Klassen, Interfaces und Pakete erstellt und sie mit den Methoden der Schnittstellen verseht. Natürlich noch ohne Methodenrümpfe bzw. mit minimalen Rümpfen. Dieses Vorgehen vereinfacht den Schnittstellenentwurf und stellt Konsistenz sicher.

Jeder Schnittstelle liegt ein Protokoll zugrunde. Das Protokoll beschreibt die Vor- und Nachbedingungen der Schnittstellenelemente. Dazu gehören die erlaubten Reihenfolgen, in denen Methoden der Schnittstelle aufgerufen werden dürfen, sowie Annahmen über Eingabeparameter und Zusicherungen über Ausgabeparameter. Das Protokoll von Modulen wird in der Modulsicht beschrieben. Dort, wo es sinnvoll ist, sollte es mit Hilfe von Zustands- oder Sequenzdiagrammen spezifiziert werden. Diese sind dann einzusetzen, wenn der Text allein kein ausreichendes Verständnis vermittelt (insbesondere bei komplexen oder nicht offensichtlichen Zusammenhängen).

Der Bezug zur konzeptionellen Sicht muss klar ersichtlich sein. Im Zweifel sollte er explizit erklärt werden. Auch für diese Sicht muss die Entstehung anhand der Strategien erläutert werden.

5 Datensicht

Hier wird das der Anwendung zugrundeliegende Datenmodell beschrieben. Hierzu werden neben einem erläuternden Text auch ein oder mehrere UML-Klassendiagramme verwendet. Das hier beschriebene Datenmodell wird u.a. jenes der Anforderungsspezifikation enthalten, allerdings mit implementierungsspezifischen Änderungen und Erweiterungen. Siehe die gesonderten Hinweise.

6 Ausführungssicht

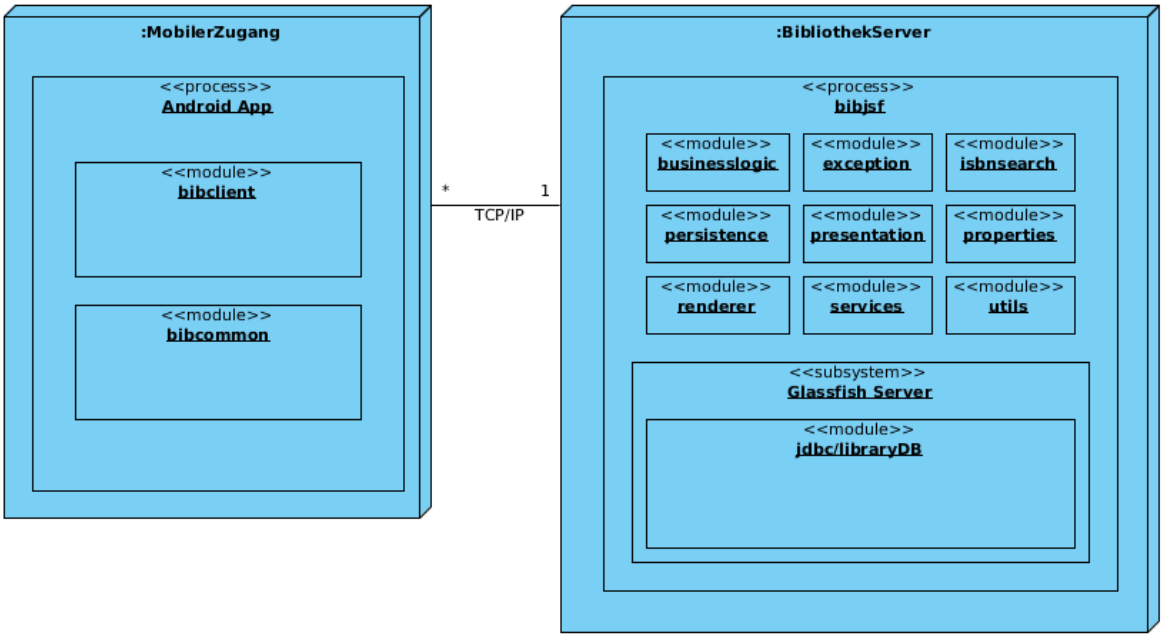
Das folgende Diagramm 10 auf der nächsten Seite zeigt das Laufzeitverhalten der Software. Auf der einen Seite haben wir den mobilen Zugang, auf dem die Android App als Prozess läuft. Die App selbst verwendet die Module `bibclient` und `bibcommon`. Von diesem mobilem Zugang kann eine TCP/IP-Verbindung zu dem Bibliotheksserver aufgebaut werden. Dabei gibt es mehrere Verbindungen zu immer nur einem Server, daher die Multiplizitäten `*` und `1`.

Auf der anderen Seite nimmt nun der Bibliotheksserver die TCP/IP-Verbindungen an. Er ist gleichzeitig Server und Datenbankserver. Die Datenbank läuft auf dem Subsystem Glassfish Server. Der Server verwendet die Module `businesslogic, exception, isbnsearch, persistence, presentation, properties, renderer, services`

und `util`.

Das komplette System läuft mit zwei Prozessen: einmal mit der Android App und dem anderen Prozess `bibjsf`, der das Subsystem mit der Datenbank enthält. Prinzipiell gibt es unendlich viele mobile Zugänge bzw. App-Prozesse, die auf einen Bibliotheksserver zugreifen.

Abbildung 10: Ausführungssicht



7 Zusammenhänge zwischen Anwendungsfällen und Architektur

In diesem Abschnitt sollen Sequenzdiagramme mit Beschreibung(!) für zwei bis drei von Euch ausgewählte Anwendungsfälleeinen von Euch ausgewählten Anwendungsfall erstellt werden. Ein Sequenzdiagramm beschreibt den Nachrichtenverkehr zwischen allen Modulen, die an der Realisierung des Anwendungsfalles beteiligt sind. Wählt die Anwendungsfälle so, dass nach Möglichkeit alle Module Eures entworfenen Systems in mindestens einem Sequenzdiagramm vorkommen. Falls Euch das nicht gelingt, versucht möglichst viele und die wichtigsten Module abzudecken.Dazu könnt ihr Euch einen Anwendungsfall heraussuchen, der möglichst viele Module der Architektur abdeckt. In SWP-2 werden wir mehrere Anwendungsfälle betrachten und eine umfangreichere Abdeckung der Architektur anstreben.

8 Evolution

In diesem Abschnitt geht es um mögliche Änderungen, Anpassungen bzw. Erweiterungen, die vorgenommen werden müssten, wenn sich Anforderungen des Systems ändern. Dabei ist wichtig, dass sich solche Änderungen möglichst modular realisieren lassen, ohne die bestehende Architektur komplett zu verändern, was sehr aufwändig und somit nicht wünschenswert wäre.
Im Folgenden werden einige wichtige mögliche neue Anforderungen bzw. Erweiterungen aufgelistet und deren jeweiligen zu implementierenden Änderungen an der Architektur beschrieben.

Erweiterungsmöglichkeiten aus der Anforderungsspezifikation

Wir haben bereits in der Anforderungsspezifikation im Punkt "Ausblick" zwei mögliche Änderungen genannt, die wir im hier detaillierter beschreiben möchten.

1. Medientypenzuwachs

Wie schon in der Anforderungsspezifikation im Punkt "2.7 Ausblick" angedeutet, wäre eine zu erwartende bzw. recht wahrscheinliche Erweiterung der Bibliothekssoftware, neben den vorhandenen Medien wie Büchern, Zeitschriften, CDs usw. weitere Medien wie z.B. Blu-Rays verwenden zu wollen und diese entsprechend im System aufzunehmen.
Da in unserem System nicht alle Medien allein für sich stehen, sondern ihre gemeinsamen Eigenschaften ihrer Klassen in einer Superklasse namens `Medium` zusammengefasst werden, ist es recht einfach, einen solchen Medientypenzuwachs ohne großen Aufwand zu realisieren. Man muss lediglich eine weitere Subklasse der Klasse `Medium`, z.B. mit Namen `BluRay` ins Modul `bibcommon` einbauen.

2. Erweiterung des Systems für iOS-Geräte

Wir werden für unser System eine Android-App entwickeln, die für den mobilen Zugang mit Smartphones einen Zugang zu dem Bibliothekssystem bietet. Da Android sehr verbreitet ist, werden wir mit dieser Lösung sicherlich viele Nutzer erreichen können, jedoch natürlich längst nicht alle.
Insofern wäre zu überlegen, ob man nicht auch eine zusätzliche App für iOS-Geräte entwickeln könnte. Um dies zu realisieren, bedarf es jedoch nicht, wie oben angestrebt, modularen Änderungen, sondern der Entwicklung eines grundlegend anderen Systems. Insofern kann an dieser Stelle auch nicht beschrieben werden, welche Änderungen vorzunehmen sind, weil es sich hierbei ja eben nicht um aufbauende Erweiterungen des bestehenden Systems, sondern im Prinzip um eine Neuentwicklung einer zweiten App handeln würde.

Zusätzliche denkbare Erweiterungen

Über die genannten Punkte aus der Anforderungsspezifikation hinaus könnten sich weitere Änderungen ergeben, die im Folgenden beschrieben werden.

1. Erweiterung des GUI-Layouts

Es wäre möglicherweise durchaus wünschenswert, wenn man als Benutzer nicht nur ein GUI-Design verwenden könnte, sondern mehrere. Dafür müsste eine Funktionalität hinzugefügt werden, um zwischen verschiedenen Benutzeroberflächen wählen zu können.
Dazu müssten entsprechende Referenzierungen von neuen GUI-Style-Änderungen mit Bilddateien stattfinden sowie für Textanpassungen die XML-Dateien im Paket `biblient/res/layout` verändert bzw. erweitert werden.

2. Mehrsprachigkeit

Gemäß den Mindestanforderungen wird das System so implementiert sein, dass die Möglichkeit besteht, mehrere Sprachen für das Bibliothekssystem zu unterstützen. Insofern wäre es denkbar, dass neben Deutsch eine weitere Sprache eingebaut werden könnte. Englisch würde sich natürlich anbieten.